Addressing Email Loss with SureMail: Measurement, Design, and Evaluation

Sharad Agarwal

Microsoft Research

Dilip A. Joseph

U.C. Berkeley

Venkata N. Padmanabhan

Microsoft Research

May 2006

Technical Report
MSR-TR-2006-67

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Addressing Email Loss with SureMail: Measurement, Design, and Evaluation

Sharad Agarwal
Microsoft Research

Dilip A. Joseph*
U.C. Berkeley

Venkata N. Padmanabhan
Microsoft Research

***Abstract*— We consider the problem of *silent* email loss in the Internet, where neither the sender nor the intended recipient is notified of the loss. Based on a detailed measurement study over three months, we find the silent email loss rate to be 0.71 to 0.91%. This suggests that the problem is non-negligible, especially since silent loss can impose a high cost.**

To address this problem, we present *SureMail*, a system that augments the existing SMTP-based email system with a notification overlay to make intended recipients aware of email they might be missing. A notification is a short, fixed-format fingerprint of an email message constructed so as to preserve sender and recipient privacy, and to prevent spoofing by spammers. Our design also avoids dependence on any special support from the email infrastructure or modifications to emails themselves, or on a PKI for email users. It also places minimal demands on users, by automating the tasks of generating, retrieving, and verifying notifications, thus only presenting valid notifications of lost email to users. Based on our prototype implementation of SureMail, we demonstrate its effectiveness in notifying recipients when they suffer silent email loss.

I. Introduction

The Internet SMTP-based email system does not guarantee the timely or even eventual delivery of messages. Email can sometimes be delayed by hours or days, or even fail to be delivered at all to the recipient(s) [12], [25]. The email sender isn't always notified when such failures occur, i.e., the message is lost without a trace, not merely bounced back or misrouted to the recipient's junk mail folder. Such *silent* failures, even if rare, impose a high cost on users in terms of missed opportunities, lost productivity, or needless misunderstanding. The SureMail system we present seeks to address this problem.

A few recent measurement studies [12], [25] have reported email loss in the range of 0.5-5%. Our measurements based on a 3-month long experiment indicate a silent loss rate of 0.71-0.91%. Besides these measurement studies, anecdotal evidence suggests that email loss is a non-negligible problem. For instance, consumer ISPs take the trouble to instruct users on what to do when email goes missing (e.g., AOL [1]) and there are companies that offer email monitoring services for businesses concerned about email loss (e.g., Pivotal Veracity [6]). Finally, the authors have themselves experienced and/or are aware of multiple instances of silent email loss recently, including that of a funding proposal and a job recommendation letter email sent to a company, a conference program committee invitation email sent from a company, a decision notification email for a major conference sent to an author in the U.S. from a server in Australia, and possible loss during a recent IMAP server upgrade at a university.

Since the current SMTP-based email system works most of the time, the approach we take in SureMail is to augment the existing system rather than replace it with a new system of uncertain reliability. SureMail provides a notification mechanism, overlaid on the unmodified SMTP email delivery system, to notify intended recipients when they are missing email. By notifying the intended recipient rather than the sender, SureMail preserves the asynchronous operation of email, together with the privacy it provides. SureMail is able to operate with the existing email infrastructure and without requiring a public key infrastructure (PKI) for email users, attributes which we believe aid real-world deployment. The additional reliability of the combination of SMTP-based email with SureMail arises from the orthogonality of the notification overlay (and hence its failure independence) with respect to the email delivery system. We have specifically designed the notification system to not be vulnerable to spam and virus-laden messages as the email system is.

This paper builds on a recent 6-page workshop position paper on SureMail [13]. While we share the same basic design as in the position paper, the novel contributions of this paper are:

- A measurement study to characterize and quantify email loss, while avoiding the shortcomings of prior measurement studies [12], [25]. (Section III)
- A refinement of the design of SureMail presented in [13], specifically to accommodate the posting of notifications by first-time legitimate senders. (Section V-G)
- Implementation and experimental evaluation of SureMail. (Section VI)

The remainder of the paper is organized as follows. We present background information on the email loss problem and discuss related work in Section II. We describe our email loss measurement study in Section III. We list the design requirements for SureMail in Section IV and present the design of the system in Section V. Section VI gives the details of our SureMail implementation and evaluation. We present a discussion of various issues pertaining to SureMail in Section VII. Section VIII concludes the paper.

II. Problem Background and Related Work

A. Nature and Extent of Email Unreliability

Email can be delayed or lost due to overload, failure, or maintenance (e.g., upgrade) of a server or relay node along the end-to-end, store-and-forward path from the sender to

*The author was an intern at MSR Redmond during the summer of 2005.

the recipient. SMTP is not an end-to-end reliable protocol. A failed relay node often does not or is not in a position to send out an indication of the failure. For example, if the node suffers a disk crash, it will likely not be able to determine which emails, if any, were lost. Ever increasing volumes of spam and email-based worm attacks make the infrastructure highly susceptible to overload and failure.

The widespread use of spam filters also contributes to email loss by sometimes causing legitimate emails to be discarded as spam. From conversations with the IT staff at a major corporation, we learned that an estimated 90% of incoming email is dropped even before it reaches user mailboxes or junk mail folders. When IP address-based white-listing or black-listing is employed, email from legitimate senders may be silently dropped if the relaying SMTP server is not on the recipient service provider's whitelist or is on its black-list. An SMTP server may be automatically dropped from a whitelist if the volume of email sent by it falls below a threshold [2]. Email sent by a traveler can be lost because he/she is forced to use his/her hotel-provided SMTP server, which may not be on the whitelists of his/her usual correspondents' service providers. Given such extensive filtering of email, it is not surprising that some legitimate email gets discarded entirely, not merely misrouted to the user's junk mail folder (we do *not* consider the latter as email "loss").

SMTP allows a server to generate non-delivery messages for emails it cannot deliver. However, the following five issues reduce the effectiveness of non-delivery messages in solving the email loss problem: (i) Emails dropped by spam filters typically do not generate non-delivery messages. (ii) Spam sent using spoofed source addresses can generate bogus non-delivery messages to the spoofed source. This can lead to general apathy toward such messages, or worse, classification of such messages as spam. (iii) Some corporations prevent generation of non-delivery messages to safeguard their privacy (e.g., to prevent an external entity from verifying if an email address is (in)valid). (iv) Servers sometimes (e.g., in the case of the disk crash noted above) do not possess enough information to generate non-delivery messages. (v) Non-delivery messages cannot warn the user about emails that are lost between the destination email server and the recipient's email client.

We are aware of two recent studies aimed at quantifying the extent of silent email loss. Afergan and Beverly [12] measure silent email loss by recording the absence of bounce-back messages for emails sent to non-existent addresses. Based on a study of 1468 mail servers across 571 domains, they found significant instances of silent email loss. For instance, 60 out of the 1468 servers exhibited a silent email loss rate of over 5%, with several others exhibiting a more modest but still non-negligible loss rate of 0.1-5%. However, a shortcoming of their methodology is that bounce-back messages may not reflect the true health of the email system for normal emails.

Lang [25] used a more direct methodology to measure email delays and losses. They used 40 email accounts across 16 domains and made direct measurements by repeatedly sending emails to these accounts over a 3-month period. They report an overall silent email loss rate of 0.69%, with the loss rate being over 4% in some cases. While the study does not depend on bounce-backs, it may be biased by the use of a single sender for all emails and the use of specially crafted "non-standard" emails (with an empty body and a message sequence number in the subject line) that could increase the likelihood of the email being filtered as spam. In Section III, we describe our measurement study that addresses some of these shortcomings. We observe a silent loss rate of 0.71-0.91%.

To put these findings in perspective, a silent loss rate in the vicinity of 0.5% corresponds to the loss of one email in 200, on average. We believe that this is a non-negligible rate of loss, especially since a user has little control over which emails are lost. Loss of important emails (like the funding proposal and recommendation letter emails mentioned in Section I) may have a serious adverse impact on users.

B. Prior Work on Addressing Email Unreliability

There have been various proposals to address the email unreliability problem, ranging from simple augmentation of the current email system to radical redesign. In the former category is the message disposition notification mechanism [17] (i.e. "read receipts"), where the sender requests the recipient to send an acknowledgment when an email has been downloaded or read. While many email clients (e.g., Microsoft Outlook, Thunderbird) support read receipts, anecdotal evidence shows that most users do not enable this feature because it exposes too much private information, viz., how often a user reads email, and whether and when the user read a particular email. It conflicts with the inherent "asynchronous" use of email. More importantly, read receipts — whether sent explicitly through the email system or implicitly through accesses to embedded web content — tell spammers whether an email account is active, thereby making their spamming more "effective".

There has been work on re-architecting email servers or the email delivery system to enhance reliability, e.g., Porcupine [31], POST [3], [26]. While improving the reliability and availability of email systems is certainly desirable, that alone will not solve the email loss problem because of spam and the resulting filtering of email. Also, while a public key infrastructure (PKI) for users, as assumed by POST, can help with the spam problem, it could be an impediment for deployment. In contrast, SureMail does not modify the underlying email delivery system and keeps the notification layer separate. This avoids the need to build (or modify) the complex functionality of an email delivery system and ensures that even in the worst case, the performance of email delivery with SureMail running on the side is no worse than with the existing email system.

There has also been much work on improving spam filtering techniques to reduce or eliminate false positives while still doing effective filtering (such as [8], [10], [7]). Although improving the accuracy is certainly useful, we believe that it is difficult to eliminate false positives entirely given that spam is constantly evolving to mimic legitimate traffic. Also, the severity of the spam problem sometimes necessitates blanket, content-independent filtering (e.g., IP address blacklisting) to reduce processing load on email servers. Hence, a notification system such as SureMail operating outside of the email system is still useful.

Finally, some prior work leverages social networks to exchange whitelist information or otherwise authenticate email senders (e.g., [15], [20]). In particular the RE: system [20] exploits friend-of-friend relationships among email correspondents to populate whitelists automatically. This is similar in spirit to the introduction mechanism in SureMail presented in Section V. However, there are a number of differences. First, RE: tries to prevent classifying email from known senders as spam (and thereby cut down on one source of email loss) whereas SureMail tries to alert users to email loss no matter what the cause. Second, RE: needs the cooperation of the domain administrators both to prevent discarding of emails by spam filters in the infrastructure and to run attestation servers. In contrast, SureMail operates outside of the email system and hence can benefit a group of cooperating users without the need for infrastructure modifications or domain participation. SureMail also leaves the emails themselves untouched, unlike RE:. Third, in terms of protocol operation, RE: incurs the overhead of a handshake between the sender and the recipient for each email whereas SureMail can choose to post notifications (and incur overhead) selectively, say depending on the importance of an email or whether it has already been replied to. Finally, SureMail leverages the practical difficulties of email eavesdropping to provide security no worse than with the current email system, thereby avoiding the need for more complex protocols.

III. Email Loss Measurement

Our design of a system to address silent email loss was motivated by our own experience with such loss. Nonetheless, we want to quantify the extent of email loss to understand the wider impact of our system. Due to privacy concerns and the logistical difficulty of monitoring email servers across many different corporations, we have to resort to sending our own emails and measuring their loss. We have designed an experiment that is similar to the one by Lang [25]. However, we improve upon that study by using multiple sending accounts and email bodies and subjects from a corpus of email user archives.

A. Experiment Setup

1) Email Accounts: To measure email loss on the Internet, we obtained several email accounts for sending email and receiving email, which are listed in Table I. We obtained accounts from academic, commercial and corporate systems. Of the non-academic systems, we included free email providers (such as GMail) and email providers that charged us for POP or IMAP access (such as Yahoo) and a private system (Microsoft corporate). Our list includes systems in different countries, including Australia, Canada, New Zealand, UK and USA. In most cases, we obtained two mailboxes - a and b [1] - to catch cases where particular accounts are misconfigured or different email accounts map to different servers. Most systems allow us POP3 or IMAP access to retrieve email, with the two exceptions using Exchange and some form of RPC over HTTP. While all accounts allowed us to retrieve emails, not all were convenient for programatically sending emails - for instance, some only allowed SMTP connections from the remote system's domain. We were unable to send emails from the Hotmail, UCLA and Cubinlab email accounts. The rest all provided an SMTP interface for sending emails, most with password protection over some form of encryption such as SSL. Overall, we have 42 email accounts for receiving email, of which 36 can send emails.

Email Addresses	Type	Receive	Send
a@microsoft.com	Exchange	√	√
{a,b}@fusemail.com	IMAP	√	√
{a,b}@aim.com	IMAP	√	√
{a,b}@yahoo.co.uk	POP3	√	√
{a,b}@yahoo.com	POP3	√	√
a@hotmail.com	HTTP	√	X
{a,b}@gawab.com	POP3	√	√
{a,b}@bluebottle.com	POP3	√	√
{a,b}@orcon.net.nz	POP	√	√
{a,b}@nerdshack.com	POP3	√	√
{a,b}@gmail.com	POP3	√	√
{a,b}@eecs.berkeley.edu	IMAP	√	√
{a,b}@cs.columbia.edu	IMAP	√	√
{a,b}@cc.gatech.edu	POP3	√	√
{a,b}@nms.lcs.mit.edu	POP3	√	√
{a,b}@cs.princeton.edu	POP3	√	√
{a,b}@cs.ucla.edu	POP3	√	X
{a,b}@cubinlab.ee.mu.oz.au	POP3	√	X
{a,b}@usc.edu	POP3	√	√
{a,b}@cs.utexas.edu	POP3	√	√
{a,b}@cs.uwaterloo.ca	POP3	√	√
{a,b}@cs.wisc.edu	IMAP	√	√

TABLE I : Email Accounts Used in Study

2) Email Content: To control the 42 accounts, we rely on programmatically sending and receiving emails, because it is difficult to obtain 42 human subjects who need to communicate with each other. However, we need to use email content that is typically sent by humans. We consider the "Enron corpus" - a large set of email messages made public during the legal investigation around the Enron corporation. In particular, we use the corpus from the "UC Berkeley Enron Email Analysis Project" [9]. This is a subset of about 1700 messages, selected for their business-related content, while trying to avoid very personal emails and spam. Of these, we use a subset of 1266 emails where

[1] We have anonymized the mailbox names. We were unable to obtain two mailboxes at microsoft.com and hotmail.com due to non-technical reasons.

each of the 1266 emails has a uniquely identifiable subject. This aids us in subsequent matching of sent emails with received emails. We consider only the body and subject of the email and ignore the rest of the header.

3) Email Attachments: Even though we use the subject and body of emails actually sent and received by real users, the corpus we consider does not have attachments. To understand the impact, if any, of attachments on email loss, we include a set of attachments of various types in our experiments. We consider the set of attachments in Table II. We include many attachment types - different image formats, different document formats and compressed archives. We also include different content types - marketing material, technical material and humorous material. We avoid large attachments (over 150KB) since we do not want to overburden the hosting email domains. The modest number and types of attachments considered in our experiment may influence the loss rate observed.

- 1. Seed random number generator
- 2. Pick a sender email address at random
- 3. Pick a receiver email address at random
- 4. Pick an email from corpus at random
- 5. Parse email and use the subject and body
- 6. Flip a coin with certain probability
- 7. If heads, pick an attachment at random
- 8. If such an email has not been sent before, send
- 9. Pick a sleep period at random and sleep
- 10. Go back to step 2

Fig. 1. Pseudo-code for Sending Process

4) System Setup: With the email accounts, email corpus and attachments, we use the sending process described in Figure 1. The sending process runs on a computer running Microsoft Windows XP SP2 which is part of a network connected to the Internet via several large ISPs. The sending process consists of a Perl program that codifies Figure 1, and a C program that handles SMTP connections to servers for sending emails. This computer also runs Microsoft Outlook 2003 which is configured to download emails from all receiving accounts.

The sending program also logs all emails sent as well as any error codes reported by the SMTP connection, if any. We use a program to feed these logs to a database on another computer running Microsoft SQL Server 2005. It also reads the received emails in Outlook 2003 via the MAPI interface and feeds them into the same database. We include emails that are in the inbox of each account, as well as any junk mail or spam folders exposed through the POP3 or IMAP interface. The program also parses the contents of any bounceback messages to determine which original email bounced. In some cases, not enough information is present in the bounceback to uniquely identify the lost email - we handle these cases separately. We issue SQL queries to match sent emails with received emails, and calculate email loss statistics. The matching is done based on the following fields : sender email address, receiver email address, subject, attachment name, date. We use a 48 hour window from the sending time to look for a matching received email. We do not use the body of the email for matching, because certain email providers, such as Yahoo, insert advertisements into the body of the email.

Sending Accounts	36
Receiving Accounts	42
Email Corpus	Enron
Unique Emails	1266
Wait Time Between Sending	2-15 secs
Attachment Probability	0.3

TABLE III : EXPERIMENT SETUP

The main features of the experiment are described in Table III. We now present the general loss statistics from the experiment as well as detailed loss statistics by subject and attachment and email account. While this experiment occurred over 2 months, in Section VI we describe results from a final third month.

B. Email Loss Statistics

Start Date	11/18/2005
End Date	01/11/2006
Days	54
Emails Sent	138944
Emails Received	144949
Emails Lost	2530
Total Loss Rate	2530/138944 = 1.82%
Bouncebacks Received	982
Matched Bouncebacks	878
Un-Matched Bouncebacks	104
Emails Lost Silently	2530-982 = 1548
Silent Loss Rate	1548/138944 = 1.11%
Hard Failures	
a@gmail.com → a@gmail.com	90
a@orcon.net.nz → a@orcon.net.nz	93
a@orcon.net.nz → b@orcon.net.nz	90
b@gmail.com → b@gmail.com	71
b@orcon.net.nz → a@orcon.net.nz	93
b@orcon.net.nz → b@orcon.net.nz	128
Total	565
Conservative Silent Loss Rate	0.71%

TABLE IV : EMAIL LOSS STATISTICS, PHASE 1

The general statistics from our experiment are in Table IV. The number of emails received is higher than the ones sent, primarily due to spam, but in some cases, also due to administrative announcements such as department announcements in the case of university email accounts. Our SQL queries for matching sent emails with received emails ignore these extraneous emails. A total of 2530 emails were lost either silently or with failed delivery attempts (bouncebacks), resulting in a total loss percentage of 1.82%. While we received 982 emails informing us of failed delivery, not all of them contained enough information for us to uniquely identify the email we sent that failed. We were unable to identify 104. Even if we include all the bouncebacks, the *silent* loss rate is 1.11%. However, certain account pairs experienced “hard failures” - i.e., no emails sent from a@orcon.net.nz to b@orcon.net.nz were received by b@orcon.net.nz. The table lists these 6 account pairs. If we remove these account pairs and calculate the

File Name	Size (B)	Type	Description	Emails Sent	Emails Lost	Loss %
			(no attachment)	96631	1062	1.1
1nag_jpg2.jpg	48634	JPEG Image	Comic book cover	1489	28	1.9
Nehru_01.jpg	21192	JPEG Image	Picture of Jawaharlal Nehru	2632	55	2.1
home_main.gif	35106	GIF Image	T-shirt design	2723	65	2.4
phd050305s.gif	65077	GIF Image	Comic strip	2905	43	1.5
ActiveXperts_Network_Monitor2.ppt	105984	MS PowerPoint	Marketing information	2935	43	1.5
vijay.ppt	48640	MS PowerPoint	Slides from technical presentation	1327	18	1.4
CfpA4v10.doc	55808	MS Word	IEEE Conference CFP	2900	43	1.5
CHANG_1587051095.doc	25088	MS Word	Book description	2711	15	0.6
SSA_PRODUCTSTRATEGY_final.doc	91136	MS Word	Marketing information	2822	33	1.2
34310344.pdf	32211	PDF	Bandwidth estimation paper	2867	25	0.9
f1040v.pdf	47875	PDF	US IRS 1040-V Form with PDF DRM	2805	42	1.5
CHANG_1587051095.zip	4987	Zip	Has CHANG_1587051095.doc	2940	19	0.6
SSA_PRODUCTSTRATEGY_final.zip	43511	Zip	Has SSA_PRODUCTSTRATEGY_final.doc	2776	64	2.3
IMC_2005_-_Call_for_Papers.htm	10377	HTML	CFP for IMC 2005	2773	32	1.2
ActivePerl_5.8_-_Online_Docs__Getting_Started.htm	24598	HTML	Perl documentation	2757	23	0.8
BBC_NEWS__Entertainment__Space_date_set_for_Scotty's_ashes.htm	32776	HTML	News article	2951	42	1.4

TABLE II : EMAIL ATTACHMENTS USED IN STUDY AND TOTAL LOSS STATISTICS

conservative silent loss rate, we arrive at 0.71% or or 983 lost emails.

We also want to understand if the type of attachment or its content influences the total loss rate. Table II presents the total email loss percentage by attachment. The majority of emails did not have attachments, given our attachment probability of 0.3. Of the rest, we do not observe a significant deviation from the total loss percentage of 1.82%. While a more extensive study needs to be conducted to fully characterize the impact of attachment types on email loss, they did not influence the outcome of our experiment. Note that we did not consider executable files and scripts for our attachments, which may potentially increase loss due to email virus and trojan scanners.

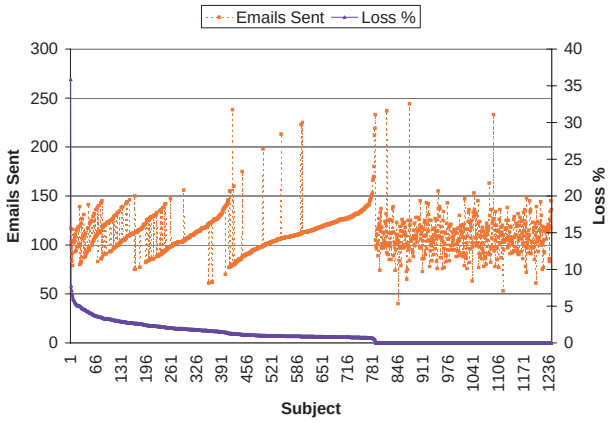

Fig. 2. Total Loss Statistics by Email Subject and Body

In Figure 2, we show the loss percentage by the email body and subject. Recall that our corpus of 1266 email bodies was selected by us such that each email has a unique subject. We sort the emails by the total loss percentage and plot them in this graph. We see that one particular email stands out with a significantly higher loss percentage - 36% loss rate for the email with the subject "CONFIDENTIAL BUSINESS PROPOSAL". We notice that this email has various properties reminiscent of the Nigerian email scam. A total of 117 emails with this subject and body were sent - if we remove this email from the total loss statistics, we are left with 2488 lost emails out of 138827 sent emails, giving us a loss percentage of 1.79%.

In this section, we presented the results of our email loss study. We found a significant total loss percentage of 1.82%, with a silent loss percentage of 0.71%. That is, out of 10,000 emails, 71 get silently lost - if a user sends about 30 emails a day, over the course of a year, over two days worth of emails get silently dropped. Even if we remove email domains with significantly higher than average loss rates or emails with significantly higher than average loss probability, the overall loss percentage does not significantly decrease. Thus we believe that a system for addressing lost email will be of significant benefit to users.

IV. DESIGN REQUIREMENTS

We list here the requirements of a solution to email unreliability that we believe will lead to the most rapid adoption.

1) **Cause minimal disruption:** The current system works for the majority of email. So rather than replace it with a new system of uncertain reliability, we should augment the system to improve its reliability. We want to inter-operate seamlessly with the existing email infrastructure (i.e., unmodified servers, mail relays, etc.), with additions restricted to software running outside the email infrastructure, e.g., on end-hosts. Users should benefit from the system without requiring any cooperation from the administrators of their email domain. In addition to minimizing disruption, we believe that these guidelines also ease the deployment of such a system.

2) **Place minimal demands on the user:** The solution should minimize demands placed on the user's time. Ideally, user interaction should be limited only to actual instances of email loss; otherwise, the user should not be involved any more than he/she is in the current email system.

3) **Preserve asynchronous operation:** The email system provides a loose coupling between senders and recipients. The sender does not know whether or when an email is downloaded or read by the recipient. Potential recipients do not know whether a sender is "online", i.e., actively sending emails to others. The proposed system should preserve such asynchronous operation, which is in contrast to other forms of communication such as telephony, instant messaging and the use of "read receipts" for email.

4) **Preserve privacy:** The solution should not reveal any more about a user's email communication behavior than the current system does. For instance, it should not be possible for a user to determine the volume or content of emails sent/received by another user, the recipients/senders of those emails, how often that user checks email, and so on. However, email, as it stands today, is vulnerable to snooping, whether on the wire or at the servers. SureMail does *not* seek to address this issue.

5) **Preserve repudiability:** It is recognized that repudiability is a key element of email and other forms of casual communication such as instant messaging [14], [11]. There is a distinction between repudiability (i.e., the ability of a sender to deny having authored an email) and forgeability (i.e., the ability of an attacker to send email purporting to be from a particular sender). The former is desirable while the latter is not, and SureMail seeks to satisfy this requirement. Note that PKI-based authentication of email users, or even authentication based on a decentralized approach like PGP, is unsuitable from the viewpoint of providing repudiability.

6) **Maintain defenses against spam and viruses:** The proposed system should not make it any easier for spam and email viruses to circumvent the defenses that are in place or make it easier for spammers to determine the validity of an email address. All email should continue to be routed through spam filters.

7) **Minimize overhead:** The proposed system should minimize the amount of overhead imposed, in terms of additional traffic and messaging. In particular, we would like to avoid duplicating the work done by the current email system on the data path, i.e., in conveying the (potentially voluminous) message bits from the sender to the recipient.

V. SureMail Design

We now describe the design of SureMail to satisfy the requirements listed above. We continue to use the current email system for message delivery. However, we augment it with a separate overlay based *notification* system. The notification system allows the intended recipient of an email to determine if he/she is missing any emails that were sent to him/her. Depending on the recipient's policy and the identity of the sender, the recipient can choose to request the sender (via email or out-of-band) to resend the lost information/email. Thus SureMail tries to assure senders that either email is delivered or the intended recipients will discover that it is missing.

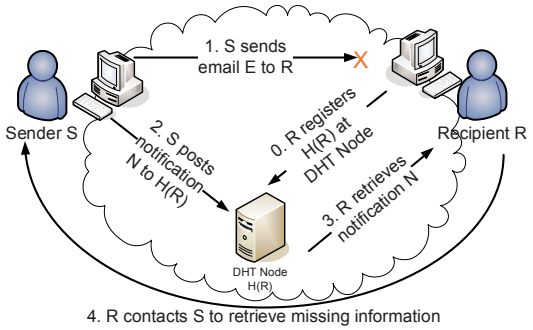

Fig. 3. Overview of SureMail

The basic idea shown in Figure 3 is as follows. When a sender sends an email to a recipient, it also stores a hash of the email contents in a DHT (Distributed Hash Table), indexed by a key derived from the recipient's email address. The recipient looks up the DHT periodically to see if there are any notifications that were sent to it. If so, the recipient compares the message hashes obtained from the DHT with hashes computed locally on messages it has received. Any hashes present in the former but not in the latter may indicate missing email. The recipient does not remove notifications from the DHT, but rather lets them expire based on a TTL.

The DHT system for notifications could be run on dedicated servers and/or on the client computers of the participating users (e.g., office computers that are mostly online). While we are agnostic to the choice of the DHT system, we assume that the DHT nodes support SureMail-specific operations, in addition to the standard $put()/get()$ operations for storing/retrieving $(key, value)$ pairs. It may be possible to leverage a system like OpenDHT [30] or Chord for this purpose. Our implementation described in Section VI is integrated with Chord.

While their scalability and availability [30], [18] makes DHTs attractive for SureMail, an alternative would be to have each organization set up dedicated servers to receive notifications for its users and advertise these servers through DNS (akin to MX records for mail servers). However, this would require cooperation of the domain administrators, which may be an impediment to initial deployment.

A key challenge is to prevent subversion of the system by bogus notifications generated by spammers or malicious DHT nodes. Indeed, notification spam would lead to user apathy and severely diminish the utility of SureMail. Our design of SureMail defends against notification spam without depending on a PKI (whether centralized or PGP-like) for email users, which eases deployment and also preserves the repudiability of email. The key idea is to leverage the implicit trust established when users exchange email. We describe the design of SureMail in more detail below.

A. Security Assumptions

We make the following security assumptions:

1) We assume that an email sent from a sender to a recipient is not available to potential attackers, except of

course if the attacker is itself the sender or the recipient. We believe that this is a reasonable assumption given the practical difficulties of eavesdropping on a remote user's email. While the email delivery infrastructure, including the routers on the network path, are in a position to eavesdrop (given that the vast majority of email today goes in the clear), we do not consider it as a potential attacker. Furthermore, if an attacker is somehow able to eavesdrop on email communication, the compromise of the user privacy is likely to be a far graver threat than the ability to subvert the SureMail notification system.

2) We do *not* assume that the notification infrastructure (e.g., the DHT nodes) is entirely trustworthy. In particular, a DHT node could try to spy on a user's email traffic (e.g., learn which senders email a recipient) or generate spurious notifications. Note that spurious notifications are a more serious problem than dropped notifications (which a malicious DHT node can always cause), since the former imposes a cognitive load on users while the latter leaves users no worse off than the current email system.

3) Although some DHT nodes may be untrustworthy, we assume that the majority of the nodes are trustworthy. Also, we assume limited collusion among the untrustworthy nodes. Although the SureMail protocols can be modified to accommodate any fixed limit on collusion (with a corresponding increase in protocol overhead), for ease of exposition we assume no collusion in our presentation.

B. Notation

Unless otherwise stated, we use S and R to represent the sender and the recipient, respectively, of an email. (We use these symbols to represent both the sending and receiving users and their email clients.) We assume that all nodes agree on a cryptographically-secure one-way hash function $H(M)$ that operates on message M to produce a short, fixed-length digest (e.g., a 20-byte digest with SHA1). We make the standard assumption that it is hard to reverse the function, i.e., retrieve M given $H(M)$. We also assume that all the nodes agree on a message authentication code (essentially a keyed hash function), $MAC(M, k)$, that operates on message M and key k to produce a short, fixed-length digest (e.g., a 20-byte digest with HMAC-SHA1). A MAC can be used with various well-known key values to generate new one-way hash functions. For example, we can define a new one-way hash function as $H'(M) = MAC(M, k')$, where k' is a well-known key that all nodes agree on. Finally, we assume that all nodes agree on a symmetric encryption function, $E(M, k)$, that operates on message M and key k. All DHT operations are represented using the syntax $func(keyID, [\cdots])$, where $keyID$ is the lookup key.

C. Basic Operation

When S sends an email M to R, it also posts a notification N into the SureMail DHT. N is a digest (e.g., $H(M)$) that uniquely identifies the email. The DHT responsible for the lookup key $H(R)$ [2] stores the notification until its expiration, as indicated by a time-to-live (TTL) T. T is specified by S when it posts the notification but is subject to a policy limit set by the DHT (to prevent storage resource exhaustion). Periodically, each email recipient, including R, retrieves the notifications intended for it from the DHT; R does so by presenting the lookup key $H(R)$. R then compares the digests contained in the notification with messages it has actually received to determine if it is missing any email.

This basic approach raises a number of issues, which we address in turn:

1) An attacker should not be able to retrieve the notifications intended for another user R. We present an email-based handshake registration procedure to ensure this.
2) An attacker should not be able to post notification spam (e.g., $H(M_S)$, where M_S is a spam message) and thereby mislead R about email it is missing. We present a reply-based shared secret scheme to identify legitimate notifications, and thereby block spam.
3) The notification N should reveal the identity of S to R but not to any attacker (e.g., the DHT node $H(R)$) who has somehow obtained N. The reply-based shared secret scheme also ensures this.
4) A legitimate first-time sender (F) who has never before communicated with R before should still be able to post notifications for R that are not mistaken for notification spam. We present an introduction mechanism to enable this.

D. Email-based Handshake for Registration

Although the notifications do not reveal message content and hence are not very sensitive, it is still undesirable for an attacker to be able to retrieve notifications intended for another user R. An attacker with access to R's notifications may learn something about the volume of emails sent to R, or even just that R is an active email recipient.

To make it hard for an attacker to retrieve R's notifications, we use a registration procedure to establish a shared secret between R and the DHT node $H(R)$. Upon system initialization, R, initiates registration by contacting the DHT node $H(R)$. To prevent an attacker from masquerading as R and registering on its behalf, DHT node $H(R)$ initiates a simple handshake via email to confirm that the node purporting to be R can in fact receive email sent to R and to defend against replay attacks. (This is a commonly-used procedure for authenticating users, for example, when they sign up for online services [21].) The challenge email sent by $H(R)$ contains the registration secret, k_R. R saves k_R for future use and responds to the node $H(R)$ (using direct communication, not email) to complete the registration.

[2] In the remainder of the paper, we use the term "node x" synonymously with "the node responsible for lookup key x".

At this stage, the DHT node $H(R)$ has reasonable assurance of R's identity and shares a secret with R (namely k_R) that it can use to authenticate future requests from R. The relatively heavy-weight email-based handshake is invoked just once, viz., at the time of initial registration. The limited use of the email-based handshake also limits the impact of email unreliability. R would have to retry its registration attempt only if the one-time challenge email is lost.

E. Decoupling Registration and Notification Posting

The DHT node $H(R)$ knows the identity of R and is in a position to monitor the volume of notifications posted for R. To prevent this, we decouple registration from the storage of notifications. While notifications are still posted to the DHT node $H(R)$, registration is performed at two (or more) other nodes, $H_1(R) = H'(H(R))$ and $H_2(R) = H''(H(R))$, such that each of $H_1(R)$ and $H_2(R)$ only holds part of the registration secret k_R. When it receives a request to retrieve the notifications corresponding to the lookup key $H(R)$, the DHT node $H(R)$ contacts $H_1(R)$ and $H_2(R)$ to authenticate the requester; $H_1(R)$ and $H_2(R)$ learn nothing about the notifications intended for R. Also, note that by construction, the node $H(R)$ finds $H_1(R)$ and $H_2(R)$ without learning the identity of R.

Under the assumption that the nodes $H(R)$, $H_1(R)$, and $H_2(R)$ are non-colluding, no single node is in a position to link R's identity with the notifications intended for R.

F. Reply-based Shared Secret

Now we turn to the problem of constructing notifications in a way that blocks notification spam and protects the identity of the sender S who posts the notification. Clearly, just posting a notification $N = H(M)$ does not suffice. Although an attacker may not directly benefit from notification spam (unlike email spam, that can generate additional traffic and business for the spammer), he/she may indirectly benefit from discrediting the notification mechanism.

We seek a solution that requires little or no human involvement. We make the observation that an email exchange between two users provides an opportunity for establishing a shared secret between them *without* requiring any additional messaging or even modifications to the emails. Furthermore, users rarely, if ever, exchange emails with a spammer.

Thus, if S sends an email M_1 to R and receives a reply M_2 from R, the SureMail software running on S can conclude that M_1 (or $H(M_1)$, for compactness) is a shared secret between S and R. The SureMail software on R remembers M_1 (or simply $H(M_1)$ and a second hash $H'(M_1)$, as explained below). It can also conclude that the user R cares about email from S, i.e., S is a legitimate sender from the viewpoint of R. We use this reply-based shared secret both to authenticate notifications posted by S and to securely convey S's identity to R.

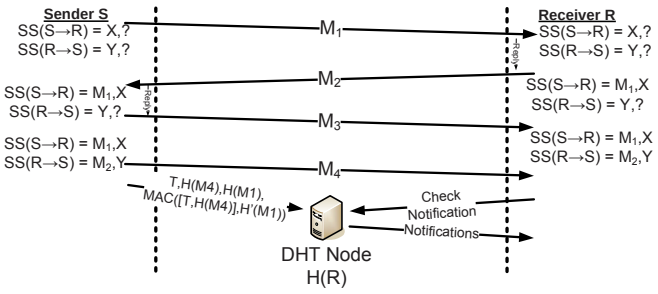

Fig. 4. Shared secret (SS) evolution between S & R in reply-based scheme

The notification for a new message, M_{new} from S to R is constructed as follows: $N = [T, H(M_{new}), H(M_1), MAC([T, H(M_{new})], H'(M_1))]$. T is the time-to-live for the notification. $H(M_{new})$ (the *bodyHash*) is the hash of the contents of the new message for which the notification is being constructed. $H(M_1)$ (the *shared secret identifier (ssId)*) implicitly identifies S to R and R alone. Basically, R can look up $H(M_1)$ in its cache and verify that it corresponds to a shared secret that it had established with S, and thus identify S. Finally, $MAC([T, H(M_{new})], H'(M_1))$ (the *authenticator*) proves to R that this is a genuine notification generated by S. Only S and R are in a position to (pre)compute the key $H'(M_1)$ used in the MAC. Possession of $H(M_1)$ alone, say from a prior notification, does not permit an attacker (e.g., the DHT node $H(R)$) to compute $H'(M_1)$ and hence post a fake notification.

Upon retrieving notifications, the SureMail software at R checks each notification to see if it is valid. If it finds any valid notifications that point to missing email, it alerts the user, presenting him/her the corresponding sender email addresses. SureMail leaves it to the user to decide the recovery action to be taken (e.g., contacting the sender(s) out-of-band). Notifications that are found to be invalid are ignored.

We now briefly discuss various issues pertaining to this reply-based shared secret scheme:

1) Shared Secret Maintenance: An email such as M_1 is used as the basis of a shared secret only if there is reason to believe that it is a secret between S and R. In SureMail, we deem a message to be a candidate for a shared secret only if R is the sole addressee on the message, a determination that can be made automatically without requiring human intervention. We believe that users do not publicly or widely share such directly addressed email (as opposed to mailing list emails that are often archived on the Web).

In terms of state, each node needs to remember two sets of shared secrets for each email address it corresponds with — one for posting notifications and the other for validating notifications it receives. The shared secrets are updated with each new email exchange, as shown in Figure 4. S remembers the hashes of all messages sent by it since the most recent one replied to by R. Likewise, R remembers the hashes of all emails from S (which it had replied to)

since the most recent one that S used as a shared secret in a notification. This constant renewing of the shared secret helps to age out old shared secrets that may have been inadvertently leaked. It also naturally supports user migration, i.e., when a user moves to a new client (say because of a change of location or computer failure) and needs to reestablish shared secrets with his/her correspondents.

2) Reply Detection: For S to conclude that M_1 is a shared secret with R, it needs to determine that a subsequent message M_2 received from R is in fact a reply to M_1. In some cases, this can be done by matching the "In-Reply-To" header field in M_2 with the "Message-ID" field of M_1. However, in other settings, the Message-ID is inserted by the server and hence is unknown to the sending client S. Nevertheless, the reply often contains remnants of the original message in the form of quoted text, which can be used for reply detection. We describe and evaluate such a heuristic in Section VI-B.

3) Comparison with PKI/PGP-based approaches: The reply-based shared secret scheme preserves the repudiability of email, which is one of the design requirements listed in Section IV. The shared secret between S and R is not meaningful to any other party. So although R can satisfy itself with the authenticity of a notification N from S, it cannot use N to prove to any other party that the message M_{new} was authored by S. In contrast, if S had a well-known public key and it were to sign the notification N with the corresponding private key, it would not be able to repudiate its authorship of M_{new}.

On the flip side, the reply-based shared secret scheme is susceptible to eavesdropping, unlike a PKI-based scheme. However, as discussed in Section V-A, eavesdropping on another user's email is difficult in practice.

G. First-time Sender

In the design of SureMail presented thus far, the sender uses a reply-based shared secret established via prior email exchange with the intended recipient to mark its notification as authentic. Consequently, a legitimate first-time sender (FTS for short) who has not exchanged email with the recipient previously would not be in a position to mark its notification as authentic. On the other hand, it is conceivable that FTS emails are more prone to being dropped by existing spam filtering systems, making it especially important that SureMail works well in the FTS scenario.

Although at first glance it might seem that a legitimate first-time sender is indistinguishable from a spammer, in practice, there are social links that may set apart a legitimate FTS from a spammer. [16] shows that email networks exhibit *small-world* properties. This implies that, although the FTS may never have communicated with the intended recipient, with high probability, he/she may have communicated with an intermediary who may have in turn communicated with the recipient. For example, such an intermediary may be a colleague of the intended recipient at the same institution. The intermediary is thus in a position to "introduce" the FTS to the recipient.

Consider an FTS, F, who is trying to communicate with R. Assume that there is an intermediary I with whom both F and R have communicated in the past (we call I a *correspondent* of F and R). Our goal is to enable F to post a notification for an email it sends R such that R does *not* discard the notification as spam. Although we wish to leverage the relationship that F and R both have with I, we would like to prevent any adverse impact on privacy. Ideally, we would like to (a) prevent I from learning that F intends to communicate with R, (b) prevent F from learning that I and R have previously communicated, and (c) prevent R from learning that F and I have previously communicated. In the event that (c) cannot be satisfied, we also consider a diluted form of (c), termed (c'), which requires that R should not learn about any of F's correspondents other than those it shares in common with F.

There has been prior work on similar problems in the context of exchanging email white-lists. LOAF [15] uses Bloom filters to exchange address books. To each email it sends to R, I attaches a Bloom filter containing addresses to which it has sent email in the past. If I trusts F, F will be included in this Bloom filter and therefore F is introduced to R through I. This scheme satisfies property (a), but not (b), (c) or (c').

RE: [20] uses the *friend-of-friend (FoF)* approach in which I can attest to R that F is not a spammer. Using a homomorphic encryption-based private matching protocol [19], RE: ensures that F does not learn the identities of R's friends and that R does not learn the identities of any of F's friends except those in common with its own friends. Thus this scheme satisfies properties (a), (b), and (c'), but not (c).

The introduction mechanism we present also satisfies properties (a), (b), and (c'), but is simpler than RE:'s mechanisms in terms of the cryptographic operations and in *not* requiring a two-way handshake between the sender and the recipient for each email.

1) SureMail Introduction Mechanism: In the SureMail introduction mechanism, each node such as I establishes a *common* secret, S_I, with all of its correspondents. Whenever I exchanges email with a new correspondent C and establishes a pairwise reply-based shared secret (S_{IC}) with it, I also conveys S_I to C. It does so by posting this information, encrypted with the pairwise shared secret S_{IC}, to the DHT node $H(C)$. Thus, all correspondents of I share a common secret, S_I. One correspondent, say F, can use S_I to authenticate notifications it may post for another correspondent, say R, even if F is a first-time sender with respect to R.

A notification from an FTS, F, for an email M that it has sent to R takes the form $N = [T, F, H(M), H(S_I), MAC([T, S, H(M)], H'(S_I))]$.
As before, this construction prevents an attacker who is in

possession of N from learning the secret S_I or constructing fake notifications. The main difference compared to the non-FTS construction presented in Section V-F is that the identity of F needs to be included explicitly, either in the clear as shown above or encrypted with the secret, S_I.

A key question is how F knows to use S_I rather than a secret S'_I obtained from another correspondent (and potential intermediary), I'. Clearly, satisfying property (b) requires that F not know whether a particular intermediary, such as I, has been a correspondent of R. We believe that it is appropriate to rely on human input, since the FTS scenario occurs relatively infrequently. When SureMail at F detects that it is an FTS with respect to the intended recipient R, it alerts the user and asks for a recommendation of one or more intermediaries from among F's correspondents. The human user can often make an informed choice, say by picking a colleague of the intended recipient, R, as the intermediary. SureMail can aid the process by automatically listing correspondents who are in the same email domain as R and hence are likely to be suitable intermediaries.

In the extreme case, F can include shared secrets obtained from all of its correspondents in the notification. R can determine if any are in common with the set of shared secrets it has obtained from its correspondents. If so, it deems the introduction as valid and honors the notification. Given that the shared secrets are opaque quantities, R does not learn anything from the shared secrets that originated from correspondents of F that are not common to R. Thus property (c') is satisfied. Note that including shared secrets from all correspondents generates notification traffic volume that is proportional to the number of correspondents. Schemes such as RE: incur a similar overhead to satisfy properties (a), (b), and (c').

2) Leakage of Shared Secret: What happens if a correspondent of I such as F passes on the secret S_I to spammers? In general, shared secrets in SureMail are set up and shared only with "trusted" peers, where trust is established based on email exchange. There is always an element of risk that a peer may not be worthy of this trust. The introduction mechanism carries the added risk that the identity of the untrustworthy peer may not be readily apparent (since all such peers hold the same secret, S_I).

We make a few remarks in this context. First, the damage done by accidental leakage can be limited by updating the shared secret periodically. Second, a recipient such as R can accord lower priority to notifications authenticated using a common shared secret such as S_I compared to those authenticated using a pairwise shared secret such as S_{IR}. Third, if property (c) is *not* satisfied, R can at least identify I or one of its correspondents as the source of the leak. If this is a frequent occurrence, the user R could be alerted to reconsider its trust of I. So in practice, property (c'), which provides a weaker assurance of privacy, may be more desirable than (c).

VI. Implementation and Evaluation

In this section, we describe our implementation of the SureMail notification system. We also repeat our email loss experiment to demonstrate the effectiveness of the notification system for catching lost emails. Since our test involved programmatically sending emails from the Enron corpus and do not have an automatic system for email reply, we do not include shared secret generation in this experiment. Instead, we separately describe our implementation of reply detection for shared secret generation and evaluate it by testing it on a corpus of emails.

A. SureMail Notification System

1) Implementation: We implemented the SureMail notification system in approximately 4000 lines of C++. The system consists of two components - *SureMailServer* and *SureMailClient*. SureMailServer exposes a *put/get* interface and runs on a single remote machine.

SureMailClient runs on a user's computer and snoops on incoming and outgoing emails without requiring any modifications to the user's email client software or the email delivery infrastructure. In our implementation with Microsoft Outlook, we snoop on emails by registering extended MAPI[24] callbacks with the email servers. For email systems (like UNIX *mail*) which store emails in text/mbox files, we could continuously monitor the files for new emails.

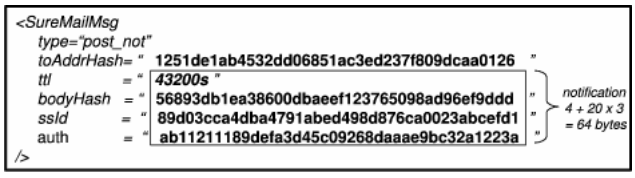

Fig. 5. Post Notification Message Format

The messages exchanged between an SureMailClient and SureMailServer are in XML format. Figure 5 shows a sample post notification message. The *bodyHash, shared secret identifier (ssId), authenticator (auth)* and *ttl* fields represent $H(M_{new})$, $H(M_{old})$, $MAC([T, H(M_{new})], H'(M_{old}))$ and T of the notification, as defined in Section V-F. $H(M_{new})$ and $H(M_{old})$ are 20-byte SHA1 hashes. $MAC([T, H(M_{new})], H'(M_{old}))$ is a 20-byte HMAC-SHA1 hash. T is 4 bytes long. The total notification size is thus 64 bytes. We used the OpenSSL[5] library for all cryptographic and hash operations. The total size of the notification message, including the overheads for the verbose XML text and the hash of the recipient email address is 166 bytes. Although a binary message format would reduce the message size to 85 bytes, we used the larger XML messages due to the flexibility provided by XML.

2) Evaluation: To evaluate SureMailClient and SureMailServer, we continue our experiment from Section III for a third month. This time, in addition to sending and

receiving emails, we also use SureMailClient in the sending process to post notifications to a centralized SureMailServer, running on a remote machine that is not in the same network as the email sending machine. We modified our sending program to also generate notification XML messages and invoke SureMailClient to post them to the SureMailServer. In this experiment, we did not use the *ssId* and *auth* fields. In addition, we had SureMailClient retrieve notifications for each receiving account every 2 hours.

Start Date	01/11/2006
End Date	02/08/2006
Days	29
Emails Sent	19435
Emails Lost	653
Total Loss Rate	653/19435 = 3.36%
Bouncebacks Received	406
Matched Bouncebacks	378
Un-Matched Bouncebacks	28
Emails Lost Silently	653-406 = 247
Silent Loss Rate	247/19435 = 1.27%
Hard Failures	
a@gmail.com → a@gmail.com	13
a@orcon.net.nz → a@orcon.net.nz	8
a@orcon.net.nz → b@orcon.net.nz	11
b@gmail.com → b@gmail.com	13
b@orcon.net.nz → a@orcon.net.nz	15
b@orcon.net.nz → b@orcon.net.nz	10
Total	70
Conservative Silent Loss Rate	0.91%
Notifications Received	19435

TABLE V : EMAIL LOSS AND NOTIFICATION STATISTICS, PHASE 2

In Table V, we present the results of this second experiment, which ran for almost a month. 19435 emails were sent and the corresponding notifications were posted. As before, bounceback messages were received, most of which were matched to sent emails and some were not. After we account for the un-matched bounceback emails and email accounts with hard failures, we arrive at a silent email loss rate of 0.91%, which is similar to our finding from Phase 1 of the experiment. Regardless of whether an email was received, lost silently or lost with a bounceback, all notification messages were received. Thus despite a 0.91% silent loss rate, the receiving accounts were notified of every loss. This demonstrates the effectiveness of our notification system for addressing email loss.

Furthermore, the overhead of our notification system is minimal, specifically with respect to generating notification messages. On a 3GHz Pentium-D computer running Linux kernel 2.6.14, notification generation for email messages of sizes 1000B, 10000B and 100000B took $94\mu s$, $832\mu s$ and $7438\mu s$ respectively (averaged over 10000 trials).

3) DHT Implementation: Despite the effectiveness of our central implementation of SureMailServer, a system for use by large numbers of users, say the entire Internet, would warrant a more scalable solution. An implementation on top of a DHT (distributed hash table) can allow the system to scale more easily. Prior work [29] has demonstrated *get()* latencies of under 200ms on average, and under 2s for the 99th percentile for the OpenDHT system running on PlanetLab.

One issue is how difficult it is to implement a notification server on top of a DHT. Since our SureMailServer exposes a *put/get* interface, we can plug in any existing DHT algorithm with little effort. We plugged in the Chord DHT implementation available from the *i3*[33] project using just 60 lines of C++. In our implementation, an SureMailClient uses a nearby SureMailServer as a first hop gateway to post and retrieve notifications from the notification overlay. This first hop SureMailServer forwards the client post/retrieve request to the SureMailServer responsible for storing the notification using the Chord DHT protocol. The response to the request is sent back to the client via the first hop SureMailServer. In ongoing work, we are evaluating SureMail on PlanetLab.

B. Reply Detection

1) Implementation: Accurate reply detection can be performed by matching the *Message-ID* and *In-Reply-To* fields of emails. However, RFC 2822[28] does not mandate these fields to be present in all messages. In most email systems[32] (for example, in the case of Microsoft Outlook mail user agents (MUA) and Microsoft Exchange mail transfer agents (MTA)), the Message-ID is inserted by the MTA and hence is unknown to the MUA sending the email. Even if inserted by the sending MUA, the Message-ID may be modified by the MTA[28]. Hence, we cannot solely rely on the Message-ID and In-Reply-To fields for reply detection.

We implemented an alternate reply detection algorithm which does not use the Message-ID and In-Reply-To fields. Our algorithm marks an email A as the reply of an email B if: (i) The subjects of A and B are identical after removing any prefixes of the form "Re:", and (ii) The bodies of A and B pass the *k-grams*[22] text similarity test. In the k-grams test, we calculate the number of common subsequences of k words (k-grams) in the bodies of emails A and B. If the number of common k-grams is greater than a threshold (M), the two emails pass the test. The k-grams of an email constitute its signature and are stored for performing reply detection tests on incoming email. In order to reduce storage space, we store 4-byte hashes of the k-grams, instead of the full k-grams.

2) Evaluation: We evaluated our reply detection algorithm on emails obtained from multiple IETF mailing list archives[4]. These email messages contained the Message-ID and In-Reply-To fields. We used these fields to automatically pair up an email and its reply, and thus generate the ground truth against which we evaluated the performance of our reply detection algorithm. We were unable to include the Enron corpus in our evaluation set as the In-Reply-To field was absent, and hence it was impossible to automatically generate the ground truth. Due to privacy concerns, we were unable to obtain another suitable non-mailing-list corpus for this analysis.

Our algorithm performed best when $k = 5$ and $M = 15$. In the 2232 emails in our evaluation set, 615 emails were replies to some other emails in the set. Our algorithm successfully detected 475 of these, yielding a false negative rate of 22.7%. Although replies often include remnants of the original email, this is not always the case, hence the false negatives. We also tested the algorithm against pairs of emails in which none of the two mails was a reply to the other. Among the 24799 pairs tested, only 47 pairs were wrongly determined as sharing a 'reply' relationship, thus yielding a very low false positive rate of 0.02%, which is desirable for SureMail. The few false positives arise from messages in the same discussion thread that include snippets of text from emails they are not direct replies to. This is unlikely to be as common with the non-mailing-list emails that would be used by SureMail's reply-based shared secret scheme. The high false negative rate implies that some replies will not be detected. This leads to (a) unnecessary posting of notifications for emails that have been replied to, and (b) missed opportunities for renewing the reply-based shared secret, both of which are not critical for our system. Hence, our reply detection algorithm is adequate for SureMail.

VII. Discussion

We briefly discuss several issues pertaining to SureMail, including enhancements to the basic design. To quantify some of our arguments, we present data derived from the analysis of the mailboxes of 15 users in a large corporation (termed the "mailbox data set" for short). These mailboxes resided on a Microsoft Exchange mail server and contained a total of 113,365 emails. This data set gives us a real-world data point on metrics of interest such as the email size distribution and the email replying behavior of users, something that the measurement experiment described in Section III is unable to provide. However, this data set is limited in size and diversity because it is difficult to recruit users due to their privacy concerns. Also, since we analyzed a snapshot of the users' mailboxes at only one point in time, we have no knowledge of emails that users either deleted or moved to offline folders outside of our reach. So we use this data set only to suggest (in)appropriateness of various design choices rather than as definitive findings.

A. Reducing DHT Overhead

Since the SMTP-based email infrastructure works fine most of the time, posting a notification in SureMail for every email sent could add up to significant and unnecessary overhead. To reduce overhead, a sender could hold off on posting a notification for a while, in the hope that an "ACK" in the form of a reply or a "NACK" in the form of a bounce-back (both detected automatically, without human involvement) would obviate the need for the notification. Holding off also provides the opportunity to coalesce notifications for multiple messages to a recipient, to further reduce overhead.

We consider the email replying behavior of users derived from the mailbox data set, based on the relevant MAPI (Messaging API) properties (e.g., time when a reply message was sent) as recorded by the Exchange server when an email is replied to. Although only 15% of incoming emails were replied to, 30% of incoming emails that were directly addressed to the user (i.e., where the user's email address was listed on the "To:" or "cc:" lines) were replied to.[3] This suggests that the implicit "ACK" provided by email replies has the potential of reducing notification workload by a non-trivial amount.

The next question is if a sender does hold off posting a notification in the hope that a reply would obviate the need for a notification, how long should it should hold off for? Based on the mailbox data set, we find that assuming that an incoming email is replied to, the median delay between the receipt of the incoming email and the reply is 3,807 seconds (i.e., just over an hour) and the 75th percentile is 36,365 seconds (i.e., just over 10 hours, which corresponds roughly to a reply within the same day or after an overnight wait). This suggests that a significant fraction of replies can be taken advantage of if we are willing to wait for one to several hours, which may be reasonable given the timescales of email communication.

Finally, to cut down the volume of notifications, the sender could decide whether to delay posting a notification or whether to post it at all based on the importance of the email. The importance could be determined automatically, say based on a preconfigured list of recipients that the sender cares about or based on a per-message indication such as the "importance" header field [23]. As a data point, only 1.78% of the emails in the mailbox data set had the importance field set to "high". So there is a significant opportunity for cutting down the volume of notifications if we decide that only the "Importance: high" emails need the additional assurance provided by SureMail. However, this assumes that in our data set, users always manually set the importance flag on all important emails.

B. When Should a User Act on Loss Notification?

A user that receives a notification that email to him/her from another user got lost is left with a dilemma - should he or she contact the sender, and if so, when? The first question will likely depend on the individual user's policy and perception of whether an email from the sender would be important. However, we can help answer the second question. In the email loss experiment from Section III, we sent and received over 130000 emails across about 40 accounts. By correlating the sent email with the received email, we can calculate the delay between sending and receiving. For the sent time, we use the actual time that our sending process initiated the outbound SMTP connection.

[3]While this fraction still seems low, a manual examination indicates that spam and bulk emails (which nevertheless are addressed directly to the user) account for many of the directly addressed emails that were not replied to.

For the received time, we use the timestamp embedded by the receiving SMTP daemon in the received email's header. Almost a third of the non-lost emails have slightly negative delays, most likely due to lack of clock synchronization between our sending machines and the various SMTP servers for our email accounts. Of the remaining emails, the median delay is 26 seconds, the mean is 276 seconds, the standard deviation is 55 minutes and the maximum is 36.6 hours. Thus we believe that if a user waits a couple of hours before acting on a notification, they can be reasonably certain that the email has been lost and not simply delayed.

C. Supporting Mailing Lists

Notifications are posted as usual for emails sent to mailing lists. The only difference is that the optimizations listed above involving holding back on notifications would not apply. The notification for an email sent to mailing list M would be sent to $H(M)$. It would be the responsibility of individual recipients to look up the DHT periodically for notifications of emails sent to mailing lists that they are members of.

D. Should emails themselves be delivered via SureMail?

Delivering emails themselves through the SureMail overlay is problematic for several reasons. First, emails are typically much larger than notifications, so transporting them through the overlay incurs a much larger overhead. For instance, the median email size (including attachments) in the mailbox data set was 3,973 bytes and the 95th percentile was 44,079 bytes.[4] In contrast, notifications are much smaller in size (85-166 bytes) as reported in Section VI. Second, it is still desirable for emails sent via the overlay to be routed through spam filters, virus scanners, etc., which leaves open the possibility of email loss. In contrast, notifications are fixed-sized, fixed-format entities that do not pose the same threat as malicious email content and hence need not be filtered the same way. Finally, using the overlay alone for delivering the emails themselves (i.e., instead of using the current SMTP-based email system) is potentially disruptive and runs the risk of under-performing the current email system.

E. Should SureMail be integrated with email infrastructure?

We consider whether SureMail's notification mechanism should be integrated directly into the existing email infrastructure, for example, by having the receiving email server or spam filter generate notifications locally for emails that are dropped.

We believe that there are several reasons why such integration could be problematic and so having SureMail operate separately would be advantageous. First, since the email server by itself cannot distinguish dropped emails originated by senders that the recipient trusts from other email (e.g., spam), the server is forced to generate notifications (including computing a hash of the message content) for *all* dropped emails. This can place a significant burden on the email server, with potentially negative impact on normal email delivery. Second, even if load were not an issue, a recipient's email server cannot by itself generate notifications that allow the recipient to distinguish between dropped emails from trusted senders from other dropped emails, resulting in burden on the user to sort through them. Avoiding this problem would require support at the sender end, as in SureMail. Third, often spam is dropped without even looking at the email content, e.g., by blacklisting certain IP addresses. According to the IT staff of a major corporation, over 90% of dropped emails are estimated to be due to such steps. It is impossible for the email server to generate meaningful notifications in such cases. Finally, to the extent that email loss happens not because of a conscious decision to drop email but due to failure, it would be hard for the (failed) infrastructure to generate meaningful notifications.

Thus we believe that keeping the SureMail notification layer separate is advantageous since it avoids burdening the email delivery infrastructure and is less prone to correlated failures of both the email delivery and notification systems.

F. What if emails could be modified?

We have designed SureMail to avoid modifying emails so as not to alter the behavior of email delivery (e.g., the likelihood of an email being classified as spam) in any way and allow users to use any email client unaltered. If we were to relax this requirement and permit user-defined email headers [28] to be inserted by customized clients, we could optimize the operation of SureMail. First, the sending client could explicit insert the message ID or an equivalent user-defined field to obviate the need for content-based reply detection (Section VI-B). Second, a notification could include the (unique) message ID instead of a hash of the message contents. Finally, the sending client could insert notifications of recently sent emails in subsequent emails, thus enabling the recipient to determine if it is missing any of those emails. However, loss detection based on such in-band notifications would work only when there is sporadic loss in an otherwise steady stream of emails. Also, in-band notifications run the risk of being lost because of email spam filtering. So an out-of-band mechanism for posting notifications is still needed as a fall-back.

G. Attacks on SureMail

Despite the steps taken by SureMail to protect the privacy of both senders and recipients, an attacker could try indirect means to glean information. For instance, a (trusted) sender could post spurious notifications with carefully controlled TTLs and then infer a recipient's email checking (or at least

[4]There is likely a bias toward smaller emails in this data set since users often file away email with large attachments in offline folders to comply with their quota on the server. So the actual distribution of the sizes of emails *sent* or *received* is likely to have been larger than these numbers indicate.

notification checking) habits based on which emails, if any, the recipient asks for a retransmission of. However, a recipient could watch out for and choose to ignore suspicious patterns of notifications (e.g., frequent notifications from a sender indicating apparent email loss).

A DHT node could use knowledge of the IP addresses from which notifications are being posted for a recipient to reverse-engineer information about the senders. This threat can be alleviated by passing the notification through a forwarding chain (as in standard DHT routing or in anonymous communication systems such as Crowds [27]) instead of directly sending it to the $H(R)$ node.

VIII. Conclusion

In this paper, we show that silent email loss is a significant problem. Our measurement study indicates that, on average, 0.71 to 0.91% of email gets lost silently, without any indication to either the sender or the intended recipient. To address this problem, we have designed and prototyped SureMail, a notification system to complement the current email infrastructure by notifying recipients about email loss. SureMail does not require any changes to the existing email infrastructure or emails themselves, and is easy for a group of users to deploy and use without requiring the cooperation of their domain administrators. SureMail includes mechanisms to defend against attackers who may seek to subvert the system by posting notification spam or intrude on the privacy of users. In our evaluation of the prototype system, SureMail ensured that the recipient was notified about every instance of loss of email sent to the recipient, thus qualitatively improving the reliability of email communication.

Acknowledgements

We want to thank all those who helped us get email accounts at their respective locations - Hari Balakrishnan (MIT), Paul Barford (Wisconsin), Constantine Dovrolis (GaTech), Ramesh Govindan (USC), S. Keshav (Waterloo), Lili Qiu (UT Austin), Jennifer Rexford (Princeton), Henning Schulzrinne (Columbia), Darryl Veitch (U Melbourne), Lixia Zhang (UCLA). We also want to thank the members of the Systems and Networking group at Microsoft Research for their feedback and for allowing us to collect information from their Exchange accounts.

References

[1] AOL Member Missing Email Self Help Page. *http://postmaster.info.aol.com/selfhelp/mbrmissing.html*.
[2] AOL Postmaster: Whitelist Information. *http://postmaster.info.aol.com/whitelist/index.html*.
[3] ePOST Serverless Email System. *http://www.epostmail.org/*.
[4] Ietf mailing lists archive. ftp://ftp.ietf.org/ietf-mail-archive/.
[5] Openssl. http://www.openssl.org.
[6] Pivotal Veracity: Tools & Intelligence for Optimizing Delivery. *http://www.pivotalveracity.com/*.
[7] Sender policy framework (spf). http://www.openspf.org/.
[8] Spamassassin. http://spamassassin.apache.org/.
[9] U.C. Berkeley Enron Email Analysis Project. http://bailando.sims.berkeley.edu/enron_email.html.
[10] Vipul's razor. http://razor.sourceforge.net/.
[11] B. Adida, S. Hohenberger, and R. L. Rivest. Fighting phishing attacks: A lightweight trust architecture for detecting spoofed emails. In *DIMACS Workshop on Theft in E-Commerce, April 2005*.
[12] M. Afergan and R. Beverly. The State of the Email Address. *ACM CCR*, Jan 2005.
[13] S. Agarwal, V. N. Padmanabhan, and D. A. Joseph. SureMail: Notification overlay for email reliability. In *ACM HotNets-IV*, Nov 2005.
[14] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use pgp. In *ACM WPES*, 2004.
[15] M. Ceglowski and J. Schachter. Loaf. http://loaf.cantbedone.org/.
[16] H. Ebel, L.-I. Mielsch, and S. Bornholdt. Scale-free topology of e-mail networks. *Physical Review E66*, Feb 2002.
[17] R. Fajman. An Extensible Message Format for Message Disposition Notifications. *RFC 2298, IETF*, Mar 1998.
[18] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing Content Publication with Coral. *NSDI*, Mar 2004.
[19] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. *EUROCRYPT*, 2004.
[20] M. J. Freedman, S. Garriss, M. Kaminsky, B. Karp, D. Mazieres, and H. Yu. Re: Reliable email. *USENIX/ACM NSDI, May 2006*.
[21] S. L. Garfinkel. Email-based identification and authentication: An alternative to PKI? *j-IEEE-SEC-PRIV*, November/December 2003.
[22] N. Heintze. Scalable document fingerprinting. In *USENIX Workshop on Electronic Commerce*, November 1996.
[23] G. Klyne and J. Palme. "rfc" 4021 - registration of mail and mime header fields. March 2005.
[24] I. D. la Cruz and L. Thaler. *Inside MAPI*. Microsoft Press, 1996.
[25] A. Lang. Email Dependability. Bachelor of Engineering Thesis, The University of New South Wales, Australia, Nov 2004. *http://uluru.ee.unsw.edu.au/~tim/dependable_email/thesis.pdf*.
[26] A. Mislove, A. Post, C. Reis, P. Willmann, P. Druschel, D. Wallach, X. Bonnaire, P. Sens, and J. Busca. POST: A Secure, Resilient, Cooperative Messaging System. *HotOS*, May 2003.
[27] M. K. Reiter and A. D. Rubin. Crowds: anonymity for Web transactions. *ACM Trans. on Information and System Security*, 1(1):66–92, 1998.
[28] P. W. Resnick. Rfc 2822 - internet message format. April 2001.
[29] S. Rhea, B. Chun, J. Kubiatowicz, and S. Shenker. Fixing the embarrassing slowness of OpenDHT on planetlab. In *Usenix WORLDS Workshop*, 2005.
[30] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. *SIGCOMM*, Aug 2005.
[31] Y. Saito, B. Bershad, and H. Levy. Manageability, Availability and Performance in Porcupine: a Highly Scalable, Cluster-Based Mail Service. *SOSP*, Dec 1999.
[32] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, 1994.
[33] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *ACM CCR*, 32(4):73–86, 2002.